

Linearly scaling direct method for accurately inverting sparse banded matrices

Pablo García-Risueño^{1,2,3}, Pablo Echenique^{1,2,3,4}

¹Instituto de Química Física Rocasolano, CSIC, Serrano 119, E-28006 Madrid, Spain

²Departamento de Física Teórica, Universidad de Zaragoza, Pedro Cerbuna 12, E-50009 Zaragoza, Spain

³Instituto de Biocomputación y Física de Sistemas Complejos (BIFI), Universidad de Zaragoza, Mariano Esquillor s/n, Edificio I+D, E-50018 Zaragoza, Spain

⁴Unidad asociada IQFR-BIFI

E-mail: garcia.risueno@gmail.com

Abstract. In many problems in Computational Physics and Chemistry, one finds a special kind of sparse matrices, called *banded matrices*. These matrices, which are defined as having non-zero entries only within a given distance from the main diagonal, need often to be inverted in order to solve the associated linear system of equations. In this work, we introduce a new $\mathcal{O}(n)$ algorithm for solving such a system, with the size of the matrix being $n \times n$. We derive analytical recursive expressions that allow us to directly obtain the solution. In addition, we describe the extension to deal with matrices that are banded plus a small number of non-zero entries outside the band, and we use the same ideas to produce a method for obtaining the full inverse matrix. Finally, we show that our new algorithm is competitive, both in accuracy and in numerical efficiency, when compared with a standard method based on Gaussian elimination. We do this using sets of large random banded matrices, as well as the ones that appear in the calculation of Lagrange multipliers in proteins.

Keywords: banded matrix - sparse matrix - inversion - Gaussian elimination Submitted

to: *J. Phys. A: Math. Gen.*

1. Introduction

In this article we present the efficient formulae and subsequent algorithms to solve the system of linear equations

$$Ax = b, \quad (1)$$

where A is a $n \times n$ matrix, x is the $n \times 1$ vector of unknowns, b is a given $n \times 1$ vector and A satisfies the equations below for known values of $m_u, m_l < n$

$$A_{I,I+K} = 0 \quad \forall K > m_u, \quad \forall I, \quad (2)$$

$$A_{I+L,I} = 0 \quad \forall L > m_l, \quad \forall I, \quad (3)$$

i. e., A is a *banded matrix* and (1) is a *banded system*. We also investigate how to solve similar systems where there are some non-zero entries not lying in the diagonal band.

Banded systems like this are abundant in computational physics and computational chemistry literature, especially because the discretization of differential equations, transforming them into finite-difference equations, often results in banded matrices [1, 2]. Many examples of this can be found in boundary value problems [3–5], in fluid mechanics [6–8], thermodynamics [9], classical wave mechanics [3], structure mechanics [10], nanoelectronics [11], circuit analysis [12], diffusion equations and Maxwell’s first-order curl equations [2]. In quantum chemistry, finite difference methods using banded matrices are used both in wavefunction formalism [13–15] and in density functional theory [16, 17]. In addition to finite-difference problems, banded systems are present in several areas, such as constrained molecular simulation [18–20], including the calculation of Lagrange multipliers in classical mechanics [21]. An important case for the calculation of Lagrange multipliers deals with molecules with angular constraints. The banded method presented here is suitable to calculate the associated Lagrange multipliers exactly and efficiently [22]. Banded matrix techniques are useful not only in linear systems, but also in linearized ones, which also appear frequently in the literature [4, 6–9, 18].

The solution of a linear system with A being a $n \times n$ dense matrix requires $\mathcal{O}(n^3)$ floating point operations[‡] (a floating point operation is an arithmetic operation, like addition, subtraction, multiplication and ratio, involving real numbers which are represented in floating point notation, the customary nomenclature in computers). However, banded systems can be solved in $\mathcal{O}(nm_um_l)$ floating point operations using very simple recursive formulae, and the explicit form of A^{-1} can be obtained in $\mathcal{O}(n^2)$ floating point operations. As mentioned earlier, there exist a number of physical problems whose behaviour is described by banded systems where $m_u, m_l \ll n$. This makes it possible to get large computational savings if suitable algorithms are used, what is even more important for computationally heavy problems like those in which the calculation of relevant quantities requires many iterations (Molecular Dynamics [14, 24], Monte Carlo simulations [25], quantum properties calculations via self-consistent field equations [26, 27], etc.).

[‡] As stated in [23], this can be reduced to $\mathcal{O}(n^{\log_2 7 \approx 2.807})$.

In this work, we introduce a new algorithm for solving banded systems and inverting banded matrices that presents very competitive numerical properties, in many cases outperforming other commonly used techniques. Additionally, we provide the explicit recursive expressions on which the algorithm is based, thus facilitating further analytical developments. The linear ($\mathcal{O}(n)$) scaling of the presented algorithm is a remarkable feature since efficiency is commonly essential in today's computer simulation of physical systems, specially in fields such as molecular mechanics [28–30] (using efficient force fields) and quantum ab initio methods [31–33].

The article is structured as follows: in section 2, we derive simple recursive formulae for the efficient solution of a linear banded system. These formulae enable the solution of (1) in $\mathcal{O}(nm_u m_l)$ floating point operations and are suitable to be used in a serial machine. In section 3, we extend these formulae to systems where some entries outside the band are also non-zero. In section 6, we briefly discuss the differences between our new algorithm and one standard method to solve banded systems. In sec. 7 we quantitatively compare the performance of both algorithms in terms of accuracy and numerical cost. For this comparison, in sec. 7.1 we use randomly generated banded systems for inputs. In section 7.2 we apply the algorithms to the problem of calculating the Lagrange multipliers which arise when imposing holonomic constraints on proteins. Finally, in section 8, we state the most important conclusions of the work. In the Appendix we provide equations for the explicit expression of the entries of A^{-1} . Some remarks on the algorithmic implementation of the methods presented here, their source code and remarks on their parallelization can be found in the supplementary material.

2. Analytical solution of banded systems

One of the most common ways of solving the linear system in equation (1) is by gradually changing the different entries of the matrix A to zero through the procedure of *Gaussian elimination* [34–36]. This procedure is based on the possibility of writing A as $A = LU$, where L is a lower triangular matrix and U is an upper triangular matrix. This way of writing A , called *LU-decomposition*, is possible (i.e., L and U exist), if, and only if A is invertible and all its leading principal minors are non-zero [37]. If one of the two matrices L or U is chosen to be *unit triangular*, i.e., with 1's on its diagonal, the matrices not only exist but are also unique.

The analytical calculations and algorithms introduced in this work are based on a different but closely related property of A , namely, the possibility of finding Q (a lower triangular matrix) and P (an upper triangular one), so that we have

$$QAP = \mathbb{I} \quad \Rightarrow \quad A^{-1} = PQ, \quad (4)$$

where \mathbb{I} is the identity matrix.

The requirements in order for these two matrices to exist are the same as those in the *LU-decomposition*, because in fact, the two propositions are equivalent. The existence of a '*QP-decomposition*' arises from the existence of the *LU* decomposition.

This is trivially proved if we make $Q = L^{-1}$ and $P = U^{-1}$. The converse implication follows from the following facts. A must be invertible so that equation (1) has a unique solution. The fact that its determinant ($\det A$) is different from zero and the relation $\det Q \det A \det P = \det \mathbb{I} = 1$ force both Q and P to have non-zero determinants and therefore to be invertible. This enables one to write $A = Q^{-1}P^{-1}$, and since the inverse of a triangular matrix is a triangular matrix of the same kind, we can identify $L = Q^{-1}$ and $U = P^{-1}$ thus proving the existence of the LU -decomposition. This equivalence also enables one to say that, as long as one of the two matrices Q and P is unit triangular, the QP -decomposition is unique.

An important qualification of this situation is that in order to solve the system in equation (1), using QP (or LU) decomposition is not the only option. We can also solve the system by performing a Gaussian elimination process that is based on the QP (or LU) decomposition of a matrix \tilde{A} , which is obtained from A by permuting its rows and/or columns. If these permutations are performed (what is called *pivoting*), the condition for $Q\tilde{A}P = \mathbb{I}$ (or $\tilde{A} = LU$) to hold is simply that A is invertible. Typically, the algorithms obtained from the pivoting case are more stable. For the sake of simplicity, derivations of this paper deal only with the non-pivoting case (the reader should notice that pivoting can be included in the debate with minor adjustments). In the supplementary material, algorithms including and lacking pivoting can be analysed.

Let us now build the matrices P and Q that satisfy (4) for a given matrix A . When we have obtained them, they can be used to compute the inverse A^{-1} , and then we will be able to solve (1). However, in this section (see also ref. [38]) we will see that there is no need to explicitly build A^{-1} , and the information needed to calculate P and Q can be used in a different way to solve (1).

We begin by writing P and Q as follows

$$P := P_1 P_2 \dots P_n = \prod_{K=1}^n P_K , \quad (5a)$$

$$Q := Q_n Q_{n-1} \dots Q_1 = \prod_{K=n}^1 Q_K , \quad (5b)$$

being

$$P_K := \begin{pmatrix} 1 & & & & & & \\ & \ddots & & & & & \\ & & 1 & & & & \\ & & & \xi_{KK} & \xi_{K,K+1} & \dots & \xi_{K,K+m_u} \\ & & & & 1 & & \\ & & & & & \ddots & \\ & & & & & & 1 & \\ & & & & & & & \ddots \\ & & & & & & & & 1 \end{pmatrix} , \quad (6)$$

and

$$Q_K := \begin{pmatrix} 1 & & & & & \\ & \ddots & & & & \\ & & 1 & & & \\ & & \xi_{K+1,K} & 1 & & \\ & & \vdots & & \ddots & \\ & & \xi_{K+m_u,K} & & & 1 \\ & & & & \ddots & \\ & & & & & 1 \end{pmatrix}, \quad (7)$$

where $K = 1, \dots, n$, and all the non-specified entries are zero. Note that P_K equals the identity matrix except in its K -th row, and Q_K equals the identity matrix except in its K -th column.

Now, the trick is to choose all coefficients ξ_{IJ} in the preceding matrices so that we have $QAP = \mathbb{I}$ in (4) (whenever the conditions for this to be possible are satisfied; see the beginning of this section).

First we must notice that given (6), multiplying a generic matrix G (by its right) by P_K is equivalent to adding the K -th column of G multiplied by the corresponding ξ coefficients to several columns of G , while at the same time multiplying the K -th column of the original matrix by ξ_{KK} :

$$(GP_K)_{IJ} = G_{IJ} \quad \text{for } J < K \text{ and } J > K + m_u, \quad (8a)$$

$$(GP_K)_{IK} = G_{IK}\xi_{KK}, \quad (8b)$$

$$(GP_K)_{IJ} = G_{IJ} + G_{IK}\xi_{KJ} \quad \text{for } K < J \leq K + m_u. \quad (8c)$$

If we take this into account, we can choose ξ_{11} so that $(AP_1)_{11} = 1$, and $(AP_1)_{1J} = 0$ for $J = 2, \dots, n$. Given the fact that A is banded (see, in particular (2, 3)), we have that

$$(AP_1)_{11} = A_{11}\xi_{11} = 1 \quad \Rightarrow \quad \xi_{11} = 1/A_{11}, \quad (9a)$$

$$(AP_1)_{1J} = A_{1J} + A_{11}\xi_{1J} = 0 \quad \Rightarrow \quad \xi_{1J} = -\frac{A_{1J}}{A_{11}}, \quad 1 < J \leq 1 + m_u. \quad (9b)$$

Operating in this way, we have ‘erased’ (i.e., turned into zeros) the superdiagonal entries§ of A that lie on its first row, and we have done this by multiplying A on the right by P_1 with the appropriate ξ_{1J} . Then, if we multiply (AP_1) on the right by P_2 and choose the coefficients ξ_{2J} in the analogous way, we can erase all the superdiagonal entries in the second row and change its diagonal entry to 1. In general, multiplying $(AP_1 \cdots P_{K-1})$ by P_K erases the superdiagonal entries of the K -th row of $(AP_1 \cdots P_{K-1})$, and turns its diagonal KK entry to 1. This procedure is called *Gaussian elimination* [37], and after n steps, the resulting matrix is a lower unit triangular matrix $A \prod_{K=1}^n P_K = AP$.

The expression for the coefficients ξ_{IJ} , with $I \leq J$ and $I > 1$ is more complex than (9a) because, as a consequence of (8a, 8b, 8c), whenever we multiply a matrix on the

§ We call *superdiagonal* entries of a matrix A to the entries A_{IJ} with $I < J$, and *subdiagonal* entries to the entries A_{IJ} with $I > J$.

right by P_K , not only is its K -th row (the one we are erasing) affected, but also all the rows below are affected (the m_l rows below in the case of a banded matrix like (4)). However, the matrix $A \prod_{L=1}^{K-1} P_L$ is 0 in all its superdiagonal entries belonging to the first $K - 1$ rows, and multiplying it on the right by P_K has no influence on these rows. Hence, the fact that we have chosen to erase the superdiagonal entries of A from the first row to the last row allows us to express the general conditions that the ξ coefficients belonging to different P_K 's must satisfy the following:

$$\left(A \prod_{K=1}^I P_K \right)_{II} = 1 , \quad (10a)$$

$$\left(A \prod_{K=1}^I P_K \right)_{IJ} = 0 \quad \text{for } I < J . \quad (10b)$$

Now, using (10a) together with (8b), we can derive the following expression for the coefficient ξ_{II} in terms of the previous steps of the process:

$$\begin{aligned} \left(A \prod_{K=1}^I P_K \right)_{II} &= \left(A \prod_{K=1}^{I-1} P_K P_I \right)_{II} = \left(A \prod_{K=1}^{I-1} P_K \right)_{II} \xi_{II} = 1 \\ \Rightarrow \quad \xi_{II} &= \frac{1}{\left(A \prod_{K=1}^{I-1} P_K \right)_{II}} . \end{aligned} \quad (11)$$

Analogously, using (10b) and (8c), we can write an explicit expression for ξ_{IJ} with $I < J$:

$$\begin{aligned} \left(A \prod_{K=1}^{I-1} P_K P_I \right)_{IJ} &= \left(A \prod_{K=1}^{I-1} P_K \right)_{IJ} + \left(A \prod_{K=1}^{I-1} P_K \right)_{II} \xi_{IJ} = 0 \\ \Rightarrow \quad \xi_{IJ} &= - \frac{\left(A \prod_{K=1}^{I-1} P_K \right)_{IJ}}{\left(A \prod_{K=1}^{I-1} P_K \right)_{II}} = - \left(A \prod_{K=1}^{I-1} P_K \right)_{IJ} \xi_{II} . \end{aligned} \quad (12)$$

Also according to (8a, 8b, 8c), for $I \leq J$

$$\left(A \prod_{K=1}^L P_K \right)_{IJ} = A_{IJ} + \sum_{M=J-m_u}^L \left(A \prod_{K=1}^{M-1} P_K \right)_{IM} \xi_{MJ} , \quad I > L . \quad (13)$$

Note that, in this equation we have $I > M$, which entails that $\left(A \prod_{K=1}^{M-1} P_K \right)_{IM}$ are subdiagonal entries. This enables one to calculate the coefficients ξ_{IJ} with $I > J$, i.e., those that correspond to the matrices Q_K , once all the coefficients in the matrices P_K have already been evaluated. We know that AP is a unit lower triangular matrix. This means that its subdiagonal I, M entry (with $I > M$) equals ξ_{IJ} , because no other changes affect this entry when multiplying AP by the different Q_K 's. If G is a generic matrix, then (7) implies

$$(Q_K G)_{IJ} = G_{IJ} \quad \text{for } I \leq K \text{ and } I > K + m_l , \quad (14a)$$

$$(Q_K G)_{IJ} = G_{IJ} + G_{KJ} \xi_{IK} \quad \text{for } K < I \leq K + m_l, \text{ for all } J . \quad (14b)$$

|| Because $I > L$ by hypothesis, and $L \geq M$.

If T is any unit lower triangular matrix (this is, its diagonal entries equal 1 and its superdiagonal entries are zero, $T_{II} = 1$, $T_{IJ} = 0$ for $I < J$), then (7) implies

$$(Q_K T)_{IJ} = T_{IJ} \quad \text{for } I \leq K \text{ and } I > K + m_l, \quad (15a)$$

$$(Q_K T)_{IJ} = T_{IJ} + T_{KJ} \xi_{IK} \quad \text{for } K < I \leq K + m_l, J \leq K, \quad (15b)$$

$$(Q_K T)_{IJ} = T_{IJ} \quad \text{for } K < I \leq K + m_l, J > K. \quad (15c)$$

Moreover, if S_{K-1} is a unit lower triangular matrix satisfying $(S_{K-1})_{IJ} = 0$ for $I > J$, $J < K$, then equations (15a, 15b, 15c) become

$$(Q_K(S_{K-1}))_{IJ} = (S_{K-1})_{IJ} \quad \text{for } I \leq K \text{ and } I > K + m_l, \quad (16a)$$

$$(Q_K(S_{K-1}))_{IJ} = (S_{K-1})_{IK} + (S_{K-1})_{KK} \xi_{IK} \quad \text{for } K < I \leq K + m_l, \quad (16b)$$

$$(Q_K(S_{K-1}))_{IJ} = (S_{K-1})_{IJ} \quad \text{for } K < I \leq K + m_l, J > K. \quad (16c)$$

Multiplying Q_1 on the left by (AP) erases the subdiagonal entries of the first column of (AP) , and multiplying $(Q_1 AP)$ on the left by Q_2 erases the subdiagonal entries of the second column of $(Q_1 AP)$. By repeating this procedure, multiplying Q_K by $(Q_{K-1} \cdots Q_1 AP)$ by the left, the subdiagonal entries of the K column of $(Q_{K-1} \cdots Q_1 AP)$ are erased. Therefore, $(Q_{K-1} \cdots Q_1 AP)$ satisfies the conditions of S_{K-1} , and hence it satisfies equations (16a, 16b, 16c). The conditions for the correct erasing are:

$$\begin{aligned} (Q_K(Q_{K-1} \cdots Q_1 AP))_{IK} &= \\ (Q_{K-1} \cdots Q_1 AP)_{IK} + (Q_{K-1} \cdots Q_1 AP)_{KK} \xi_{IK} &= 0 \quad \text{for } K < I \leq K + m_l. \end{aligned} \quad (17)$$

The expressions in (17) can be simplified. Equation (16c) implies

$$\begin{aligned} (Q_{K-1} \cdots Q_1 AP)_{IK} &= (Q_{K-1}(Q_{K-2} \cdots Q_1 AP))_{IK} = \\ (Q_{K'}(Q_{K'-1} \cdots Q_1 AP))_{I, K'+1} &= (Q_{K'-1} \cdots Q_1 AP)_{I, K'+1} = \\ (Q_{K-2} \cdots Q_1 AP)_{IK}, \end{aligned} \quad (18a)$$

where we have defined $K' := K - 1$. By repeating operations like this, it is easy to obtain

$$(Q_{K-1} \cdots Q_1 AP)_{IK} = (AP)_{IK} \quad \text{for } K < I \leq K + m_l, \quad (19a)$$

$$(Q_{K-1} \cdots Q_1 AP)_{KK} = (AP)_{KK} = 1. \quad (19b)$$

In addition, we must consider that (8a, 8b, 8c) imply

$$\left(A \prod_{L=1}^{M-1} P_L \right)_{IM} = \left(A \prod_{L=1}^n P_L \right)_{IM} / \xi_{MM} = (AP)_{IM} / \xi_{MM} \quad \text{for } I > M. \quad (20)$$

Using (19a, 19b, 20) into (17) we obtain

$$\left(A \prod_{L=1}^{M-1} P_L \right)_{IM} = -\xi_{IM} / \xi_{MM} \quad \text{for } I > M. \quad (21)$$

If we apply this on the right hand side of (13), then insert the resulting expression with $J = I$ and $L = I - 1$ into (11) and also insert it with $L = I - 1$ into (12), we get

the following recursive equations

$$\xi_{II} = \left(A_{II} - \sum_{M=I-m_u}^{I-1} \frac{\xi_{IM}}{\xi_{MM}} \xi_{MI} \right)^{-1}, \quad (22a)$$

$$\xi_{IJ} = \xi_{II} \left(-A_{IJ} + \sum_{M=I-m_u}^{I-1} \frac{\xi_{IM}}{\xi_{MM}} \xi_{MJ} \right) \quad \text{for } I < J. \quad (22b)$$

Following an analogue procedure, we obtain

$$\xi_{IJ} = \xi_{II} \left(-A_{IJ} + \sum_{M=J-m_l}^{J-1} \frac{\xi_{IM}}{\xi_{MM}} \xi_{MJ} \right) \quad \text{for } I > J. \quad (23)$$

For efficiency reasons, we prefer to define $\chi_{IJ} := \xi_{IJ}/\xi_{II}$ for $I > J$. This makes (22a, 22b, 23) become

$$\xi_{II} = \left(A_{II} - \sum_{M=I-m_u}^{I-1} \chi_{IM} \xi_{MI} \right)^{-1}, \quad (24a)$$

$$\xi_{IJ} = \xi_{II} \left(-A_{IJ} + \sum_{M=I-m_u}^{I-1} \chi_{IM} \xi_{MJ} \right) \quad \text{for } I < J, \quad (24b)$$

$$\chi_{IJ} = -A_{IJ} + \sum_{M=J-m_l}^{J-1} \chi_{IM} \xi_{MJ} \quad \text{for } I > J. \quad (24c)$$

The three equations above can be further modified with the aim of improving the numerical efficiency of the algorithms derived from them. The starting point for the summations in (24a) must be the value of M such that both ξ_{IM} and ξ_{MJ} are non-zero. We must take into account that in a banded matrix the number of non zero entries above and on the left of the I, J entry depends on the values of I, J :

- There are $m_u + (I - J)$ non-zero entries immediately above A_{IJ} .
- There are $m_l - (I - J)$ non-zero entries immediately on the left of A_{IJ} .

These properties are also satisfied in $A \prod_{L=1}^K P_L$ for all K . Therefore, if we define

$$\begin{aligned} \mu_{IJ} &:= \min\{m_u + (I - J), m_l - (I - J)\}, \\ \mu' &:= \min\{m_u, m_l\}, \end{aligned}$$

we can re-express (24a, 24b, 24c) as

$$\xi_{II} = \left(A_{II} - \sum_{M=\max\{1, I-\mu'\}}^{I-1} \chi_{IM} \xi_{MI} \right)^{-1}, \quad (25a)$$

$$\xi_{IJ} = \xi_{II} \left(-A_{IJ} + \sum_{M=\max\{1, I-\mu_{IJ}\}}^{I-1} \chi_{IM} \xi_{MJ} \right) \quad \text{for } I < J, \quad (25b)$$

$$\chi_{IJ} = -A_{IJ} + \sum_{M=\max\{1, J-\mu_{IJ}\}}^{J-1} \chi_{IM} \xi_{MJ} \quad \text{for } I > J. \quad (25c)$$

In the restricted but very common case in which $m_l = m_u =: m$, the previous equations become

$$\xi_{II} = \left(A_{II} - \sum_{M=\max(1, I-m)}^{I-1} \chi_{IM} \xi_{MI} \right)^{-1}, \quad (26a)$$

$$\xi_{IJ} = \xi_{II} \left(-A_{IJ} + \sum_{M=\max\{1, J-m\}}^{I-1} \chi_{IM} \xi_{MJ} \right) \text{ for } I < J, \quad (26b)$$

$$\chi_{IJ} = -A_{IJ} + \sum_{M=\max\{1, I-m\}}^{J-1} \chi_{IM} \xi_{MJ} \quad \text{for } I > J. \quad (26c)$$

If the matrix A is symmetric ($A_{IJ} = A_{JI}$), we can avoid performing many operations simply by using

$$\chi_{IJ} = \xi_{JI} / \xi_{JJ}, \quad \text{for } I > J, \quad (27)$$

instead of (25c). Equation (27) can easily be obtained from (25a) by induction.

The reader must also note that, although the coefficients ξ_{IJ} have been obtained by performing the products $\prod_{K=n}^1 Q_K A \prod_{L=1}^n P_L$ in a certain order, they are independent of this choice. Indeed, if we take a look to expressions (5a), (5b), (6), and (7), we can see that the K -th row (or column) is always erased before the $(K+1)$ -th one. It does not matter if we apply first Q_K or P_K to erase the K row (or column); the result of the operation will be the same. In both cases, $-G_{IK}G_{KJ}/G_{KK}$ (where $G := \prod_{M=K-1}^1 Q_M A \prod_{L=1}^{K-1} P_L$) is added to all the entries of G_{IJ} such that $I \in \{K+1, \dots, K+m_l\}$ and $J \in \{K+1, \dots, K+m_u\}$. This is valid when both the K -th row and the K -th column are not erased yet. If one of them is already erased, erasing the other has no influence on G_{IJ} with $I \in \{K+1, \dots, K+m_l\}$ and $J \in \{K+1, \dots, K+m_u\}$. In both cases $\xi_{IK} = -G_{IK}/G_{KK}$ for $I > K$, and $\xi_{KJ} = -G_{KJ}/G_{KK}$ for $J > K$. This is because all the previous rows (or columns) have been nullified before, and adding columns (or rows) has no influence on the K -th one.

Now, the algorithm to solve (1) can be divided into three stages (in our implementation we join together the first and second ones). Since $A^{-1} = PQ$ (4), these steps are:

- (i) To obtain the coefficients ξ .
- (ii) To obtain the intermediate vector $c := Qb$.
- (iii) To obtain the final vector $x = Pc$.

Now, using the results derived above, let us calculate the expressions for the second and third steps:

Whenever we multiply a generic $n \times 1$ vector v on the left by Q_K (see (7)), we modify its K -th to $(K+m_l)$ -th rows in the following way:

$$(Q_K v)_I = v_I \quad \text{for } I \leq K, \quad I > K+m_l, \quad (28a)$$

$$(Q_K v)_I = v_I + \xi_{IK} v_K \quad \text{for } K < I \leq K+m_l. \quad (28b)$$

Since $Q := Q_n Q_{n-1} \dots Q_1$, using the expression for each of the Q_K in (7), and the fact that $\xi_{IJ} = 0$ for $I > J + m_l$, we have

$$Q_{IJ} = 0 \quad \text{for } I < J, \quad (29a)$$

$$Q_{II} = 1, \quad (29b)$$

$$Q_{IJ} = \sum_{M=\max\{I-m_l, 1\}}^{I-1} \xi_{IM} Q_{MJ} \quad \text{for } I > J. \quad (29c)$$

where the maximum in the lower limit of the sum accounts for boundary effects and ensures that M is never smaller than 1.

From these relations between the entries of Q , we can get the components c_I of the intermediate vector c in the second step above:

$$\begin{aligned} c_I &= \sum_{J=1}^n Q_{IJ} b_J = \sum_{J=1}^I Q_{IJ} b_J = b_I + \sum_{J=1}^{I-1} \left(\sum_{M=\max\{I-m_l, 1\}}^{I-1} \xi_{IM} Q_{MJ} \right) b_J \\ &= b_I + \sum_{M=\max\{I-m_l, 1\}}^{I-1} \xi_{IM} \sum_{J=1}^{I-1} Q_{MJ} b_J = b_I + \sum_{M=\max\{I-m_l, 1\}}^{I-1} \xi_{IM} c_M \\ &= b_I + \sum_{M=\max\{I-m_l, 1\}}^{I-1} \chi_{IM} \xi_{MM} c_M. \end{aligned} \quad (30)$$

In the first row of the equation above, we applied (29a), and then (29c). In the second row of the equation above, we performed a feedback in the equation.

We will now turn to the third and final step of the process, which consists of calculating the final vector $x = Pc$. Whenever we multiply a generic $n \times n$ matrix G on the left by P_K (see equation (6)), the resulting matrix is the same as G in all its rows except for the K -th one, which is equal to a linear combination of the first $m_u + 1$ rows below it:

$$(P_K G)_{IJ} = G_{IJ} \quad \text{for } I \neq K, \quad (31a)$$

$$(P_K G)_{KJ} = \sum_{L=K}^{\min\{K+m_u, n\}} \xi_{KL} G_{LJ}, \quad (31b)$$

where the minimum in the upper limit of the sum accounts for boundary effects and ensures that $K + L$ is never larger than n .

If we now use the relations above to construct P as in (5a), i.e., we first take P_n and multiply it on the left by P_{n-1} , then we multiply the result, $P_{n-1}P_n$, on the left by P_{n-2} , etc., we arrive to:

$$P_{II} = \xi_{II}, \quad (32a)$$

$$P_{IJ} = \sum_{K=I+1}^{\min\{I+m_u, n\}} \xi_{IK} P_{KJ} \quad \text{for } I < J, \quad (32b)$$

$$P_{IJ} = 0 \quad \text{for } I > J, \quad (32c)$$

meaning that every row of P is a linear combination of the following rows, plus a term in the diagonal.

These expressions allow us to obtain the last equation that is needed to solve the linear system in (1):

$$\begin{aligned}
 x_I &= \sum_{J=1}^n P_{IJ} c_J = \sum_{J=I}^n P_{IJ} c_J = \xi_{II} c_I + \sum_{J=I+1}^n \left(\sum_{K=I+1}^{\min\{I+m_u, n\}} \xi_{IK} P_{KJ} \right) c_J \\
 &= \xi_{II} c_I + \sum_{K=I+1}^{\min\{I+m_u, n\}} \xi_{IK} \sum_{J=I+1}^n P_{KJ} c_J \\
 &= \xi_{II} c_I + \sum_{K=I+1}^{\min\{I+m_u, n\}} \xi_{IK} x_K .
 \end{aligned} \tag{33}$$

Now, we can use expressions (25a), (25b), and (25c) in order to obtain the coefficients ξ , and then plug them into (30) and (33) in order to finally solve (1).

To conclude, let us focus on the computational cost of this procedure. From (25a), (25b), and (25c), it follows that obtaining the coefficients ξ requires $\mathcal{O}(n)$ floating point operations. Being more precise, the summations in (25a), (25b), and (25c), require μ_{IJ} products and $\mu_{IJ} - 1$ additions ($2\mu_{IJ} - 1$ floating point operations)¶. If, without loss of generality, we consider $m_u \geq m_l$, it is easy to check that the following computational costs hold:

- Obtaining one diagonal ξ_{II} takes $2\mu' + 2 \simeq 2m_u$ floating point operations.
- Obtaining one superdiagonal coefficient ξ_{IJ} (where $I < J$) takes about $2 \min\{m_l, m_u - (J - I)\}$ floating point operations. Hence, in order to obtain all the coefficients in a column above the diagonal, there are two sets of ξ 's that require a different number of operations. The lower one requires $2m_l$ floating point operations and the upper one requires $2(m_u - (J - I))$ floating point operations. All in all, obtaining $\xi_{J-I, J}$ for $I = 1, \dots, m_u$ takes about $m_l(2m_u - m_l)$ floating point operations.
- In order to obtain the ξ_{IJ} coefficient in a subdiagonal row ($I > J$), the number of floating point operations to be performed is $\min\{m_u, m_l - (I - J)\} = m_l - (I - J)$; this is performed in such a way that the total number of floating point operations related to this row is approximately m_l^2 .

Finally, obtaining all coefficients ξ would require slightly less operations (due to the boundary effects) than $2nm_u m_l$ floating point operations. Once they are known (or partly known during the procedure to get them), we can obtain the solution vector x using the simple recursive relationships presented in this section at a cost of $4n(m_u + m_l)$ floating point operations.

¶ The meaning of m_u, m_l can be noticed in (2, 3).

3. Banded plus sparse systems

A slight modification of the calculations presented in the previous section is required to tackle systems where not all the non-zero entries are within the band. The resulting modified procedure is described in this section.

If we have

$$A' := A + \sum_{T=1}^{T_{max}} A'_{R_T S_T} \Delta^{R_T S_T}, \quad (34)$$

with A banded (see eqs. (2) and (3)) and the matrix $\Delta^{R_T S_T}$ consisting of entries $(\Delta^{R_T S_T})_{IJ} = \delta_{I, R_T} \delta_{J, S_T}$, δ_{IJ} being the Kroenecker delta, we shall say that A' is a *banded plus sparse* matrix, and

$$A'x = b \quad (35)$$

a *banded plus sparse* system. We call an *extra-band entry* any nonzero entry which does not lie in the band (this is, A'_{IJ} is an extra-band entry if it is not zero and $I < J$, $J > I + m_u$ or $I > J$, $J > J + m_l$).

In the pure banded system (section 2) only $\xi_{K, K+J}$ and $\xi_{K+I, K}$ with $K = 1, \dots, n$; $J = 1, \dots, m_u$; $I = 1, \dots, m_l$ had to be calculated. In this case, we also need to obtain

$$\begin{aligned} \xi_{IS_T} & \quad \text{if } R_T < S_T, \text{ for } I = R_T, R_T + 1, \dots, S_T - m_u - 1, \\ \xi_{R_T J} & \quad \text{if } R_T > S_T, \text{ for } J = S_T, S_T + 1, \dots, R_T - m_l - 1, \end{aligned}$$

with $T = 1, \dots, T_{max}$.

As seen in the previous section, in order to erase (i.e., turn to 0) entry G_{IJ} , with $I < J$, of a generic matrix G , we can multiply it by a matrix P_I (see (6, 8a, 8b, 8c)). This action adds the column I (times given numbers) of matrix G to other columns of G . This erases G_{IJ} , but (in general) adds nonzero numbers to the entries below it (KJ entries with $K > I$). Therefore, if these entries were zero before performing the product GP_I , they will in general be nonzero after it. This implies that they will also have to be erased. Hence, erasing the extra-band entry IJ of A' will not suffice; the entries $I+1, J$, $I+2, J, \dots, J - m_u + 1, J$ will also have to be erased. If the extra-band entry to erase A'_{IJ} is below the diagonal ($I > J$), then the Q_J matrices (7, 14a, 14b) can be used to this end, since they add rows when multiplied by a generic matrix. Erasing A'_{IJ} will probably make that entries A'_{IK} with $K = J+1, \dots, I - m_l - 1$ become nonzero, and these entries will have to be also erased.

In order to erase the extra-band entries, the expressions presented in the previous section can be used. All extra-band entries can lie in an extended band wider than the original band. But, for the sake of efficiency, the entries in the extended band which are zero during the erasing procedure must not enter the sums for the coefficients ξ, χ .

We define

$$\nu_{R_T I} := \max\{R_T, I - m_l\}, \quad (36a)$$

$$\rho_{S_T J} := \max\{S_T, J + m_u\}. \quad (36b)$$

If $R_T < S_T$ (superdiagonal extra-band entry), in addition to coefficients appearing in (25a, 25b, 25c) we have to calculate

$$\xi_{R_T S_T} = -\xi_{R_T R_T} A'_{R_T S_T} , \quad (37a)$$

$$\xi_{IJ} = \xi_{II} \left(\sum_{M=\nu}^{I-1} \chi_{IM} \xi_{MJ} \right) \text{ for } R_T < I < S_T - m_u ; \quad (37b)$$

and if $R_T > S_T$ (subdiagonal extra-band entry), in addition to coefficients appearing in (25a, 25b, 25c) we have to calculate

$$\xi_{R_T S_T} = -A'_{R_T S_T} , \quad (38a)$$

$$\chi_{IJ} = \sum_{M=\rho}^{J-1} \xi_{IM} \chi_{MJ} \quad \text{for } S_T < J < I - m_l . \quad (38b)$$

The coefficients appearing in (37a, 37b, 38a, 38b) arise from merely applying equations (25a, 25b, 25c) and avoiding to include in them the coefficients ξ , χ which are zero due to the structure of A' .

Equations (37a, 37b, 38a, 38b) have to be modified for $I < J$ if there exist $A'_{R_x S_T}$ with $R_x < R_T$. This is because erasing the upper non-zero entries by adding columns creates new non-zero entries below them, and the new relations must take this into account. Analogous corrections must be done for $I > J$ if there exist $A'_{R_T S_x}$ with $S_x > S_T$. The general rule to proceed in sparse plus banded systems is to apply equations (25a, 25b, 25c) using the maximum m'_u , m'_l so that all the nonzero entries of A' lie within the enhanced band (given by m'_u , m'_l), and avoid that the coefficients (ξ , χ) which are zero take part in the sums. The coefficients ξ_{KL} which are zero are those given by the following rules:

- If $K < L$, $\xi_{KL} = 0$ if $A'_{ML} = 0$ for $M = 1, \dots, K$
- If $K > L$, $\xi_{KL} = 0$ if $A'_{KM} = 0$ for $M = 1, \dots, L$

The computational cost of solving banded plus sparse systems scales with n , as long as the number of columns above the band and rows below it containing non-zero entries $A'_{R_T S_T}$ is small ($\ll n$). The example code for an algorithm for sparse plus banded systems can be found in the supplementary material; the performance of this algorithm is presented in sec. 7.2.

4. Algorithmic implementation

Based on the expressions (25a, 25b, 25c, 30, 33) derived in the paper, we have coded several different algorithms that efficiently solve the linear system in (1). The difference between the method in this paper and the most commonly used implementation of Gaussian elimination techniques, such as the ones included in LAPACK [34], Numerical Recipes in C [35], or those discussed in ref. [36] is that these methods perform an LU factorization of the matrix A , and the coefficients ξ for the Gaussian elimination are

obtained in several steps, whereas the method introduced here does not perform such an LU factorization, and it obtains the coefficients ξ in a single step.

In order to obtain the solution of (1) we need to get the coefficients ξ for Gaussian elimination as explained in section 4. That is, one diagonal coefficient for each row/column, plus m_u coefficients in each row and m_l coefficients in each column (except for the last ones, where less coefficients have to be calculated). More accuracy in the solution is obtained by pivoting, i.e., altering the order of the rows and columns in the process of Gaussian elimination so that the *pivot* (the element temporarily in the diagonal and by which we are going to divide) is never too close to zero. Double pivoting (in rows *and* columns) usually gives more accurate results than partial pivoting (in rows *or* columns). However, the former is seldom preferred for banded systems, since it requires $\mathcal{O}(n^2)$ operations, while the latter requires only $\mathcal{O}(n)$. In the implementations described in this section, we have chosen to perform partial pivoting on rows, as in refs. [34, 35]. In the same spirit, and in order to save as much memory as possible, we store matrices by diagonals (see [35]).

We proceed as follows: For each given I , we obtain ξ_{II} (using (25a)), and then ξ_{JI} (using (25c)) for $J = I + 1, \dots, I + m_l$. If $|\xi_{JI}| > |\xi_{II}|$, we exchange rows I and J in the matrix A and in the vector b . This is called partial pivoting in rows, and it usually gives greater numerical stability to the solutions; in our tests of section 6 the error was lowered in two orders of magnitude by partial pivoting. Next, we calculate ξ_{IJ} (using (25b)) for $J = I + 1, \dots, I + m_u$. When we have calculated all the relevant coefficients ξ for a given I , we calculate c_I using (30). We repeat these steps for all rows I , starting by $I = 1$ and moving one row at a time up to $I = n$. This ordering enables us to solve the system using eqs. (25a), (30), (33), because the superdiagonal ξ_{IJ} (i.e., those with $I < J$) only require the knowledge of the coefficients with a lower row index I , while the subdiagonal coefficients ξ_{IJ} with $I > J$ only require the knowledge of coefficients with a lower column index. We have additionally implemented a procedure to avoid performing dummy summations (i.e., those where the term to add is null), which eliminates the need for evaluating μ_{IJ} in every step. According to the pivotings performed before starting to calculate a given ξ , a different number of terms will appear in the summation to obtain it. This procedure uses the previous pivoting (i. e., row exchanging) information and determines how many ξ coefficients have to be obtained in any row or column, and how many terms the summation to obtain them will consist of (this procedure is not indicated in the pseudo-code below for the sake of simplicity). The final step consists of obtaining x from b using (33).

The pseudo-code of the algorithm can be summarized as follows:

```
// Steps 1 and 2: Calculating the coefficients  $\xi$  and the vector  $c$ 
for ( $K = 1, K \leq n, K++$ ) do
    // Calculating the diagonal  $\xi$ 's:
     $\xi_{KK} = 1/(A_{KK} + \sum_{M=K-\mu'}^{K-1} \xi_{KM}\xi_{MK})$ 
    // Calculating the subdiagonal  $\xi$ 's:
```

```

for ( $I = K + 1$ ,  $I \leq I + m_l$ ,  $I++$ ) do
     $\xi_{IK} = -A_{IK} + \sum_{M=K-\mu_{IK}}^{K-1} \xi_{IM} \xi_{MK}$     for  $I > J$ 
end for

// Pivoting:
if  $\exists |\xi_{JI}| > |\xi_{II}|$  for  $J = I + 1, \dots, I + m_l$  then
    for ( $K = 1$ ,  $K \leq n$ ,  $K++$ ) do
         $A_{IK} \leftrightarrow A_{JK}$ 
    end for
     $b_I \leftrightarrow b_J$ 
end if

// Calculating the superdiagonal  $\xi$ 's:
for ( $J = K + 1$ ,  $J \leq K + m_u$ ,  $J++$ ) do
     $\xi_{KJ} = -\xi_{KK} A_{KJ} + \sum_{M=K-\mu_{KJ}}^{K-1} \xi_{KM} \xi_{MJ}$ 
end for

// Calculating  $c_K = (Qb)_K$ :
 $c_K = b_K$ 
for ( $L = K - m_l$ ,  $L \leq K - 1$ ,  $L++$ ) do
     $c_{K+} = \xi_{K,L} c_L$ 
end for
end for

// Step 3: Calculating  $x_K = (Pc)_K = (PQb)_K$ :
for ( $K = n$ ,  $K \geq 1$ ,  $K--$ ) do
     $x_K = \xi_{KK} c_K$ 
    for ( $L = K + 1$ ,  $L \leq K + m_u$ ,  $L++$ ) do
         $x_{K+} = \xi_{K,L} x_L$ 
    end for
end for

```

In the actual computer implementation we split the most external loop into three loops ($I = 1, \dots, 2m$, $I = 2m + 1, \dots, n - 2m$, and $I = n - 2m + 1, \dots, n$), because the summations to obtain the coefficients ξ lack some terms in the initial and final rows. We store A by diagonals in a $n \times (2m_u + m_l + 1)$ matrix in order to save memory and, with the same objective, we overwrite the original entries A_{IJ} with the calculated ξ_{IJ} for $I \leq J$, and we store the ξ_{IJ} with $I > J$ in another $n \times (2m_l)$ matrix.

One possible modification to the algorithm presented above is to omit the pivoting. This usually leads to larger errors in the solution, but results in important computational savings. It can be used in problems where computational cost is more important than achieving a very high accuracy. In any case, one must note that the accuracy of the

algorithm is typically acceptable without pivoting, so in many cases no pivoting will be necessary.

In (33) we can see that no subdiagonal coefficients (ξ_{IJ} with $I > J$) are needed to obtain x from c . In (30), we can see that only ξ_{IK} are necessary in order to obtain c_I , thus making it unnecessary to know ξ_{LK} for $L < I$. Therefore, we can get rid of them once c_I is known. Since we calculate c_I immediately after calculating all ξ_{IK} , we can overwrite $\xi_{I+1,K}$ on the memory position of ξ_{IK} . If we do so, about one third of the memory is saved, since less coefficients must be stored, however, according to some preliminary tests, this option is also 20% slower than the simpler one in which all coefficients are stored independently.

It is also worth remarking at this point that the present state of the algorithm is not yet completely optimized at the low level and, therefore, it cannot be directly compared to the thoroughly optimized routines included in commonly used scientific libraries such as LAPACK [34]. This further optimization will be pursued in future works.

5. Parallelization

There exist many works in the literature aiming at parallelizing the calculations needed to solve a banded system [1, 10, 39–50]. The decision about which one to choose, and, in particular, which one to apply to the algorithms presented in this work depends, of course, on the architecture of the machine in which the calculations are going to be performed. The choice is additionally complicated by the fact that, normally, only the number of floating point operations required by each scheme is reported in the articles. However, the number of floating point operations is known to be a poor measure of the real wall-clock performance of computer algorithms and, especially, parallel ones, for a number of reasons:

- Not all the floating point operations require the same time. For example, in currently common architectures, a quotient takes 4 times as many cycles as an addition or a product.
- A floating point operation usually requires access to several positions of memory. Each access is much slower than the floating point operation itself [51]. Moreover, the number of memory accesses does not need to be proportional to the number of floating point operations.
- Transferring information among nodes in a cluster is commonly much slower than accessing a memory position or performing a floating point operation [51].

Despite these unavoidable complexities and the fact that rigorous tests should be made in any particular architecture, two parallelizing schemes seem well suited for the method presented in this work: the one in ref. [41] for shared-memory machines and the one in ref. [10] for distributed-memory machines. The former is faster if the communication time among nodes tends to zero, whereas the latter tackles the

communication time problem by significantly reducing the number of messages that need to be passed.

6. Differences with Gaussian elimination

In order to assess the performance of the method derived in the previous sections, we will present results of numerical tests of real systems. In sec. 7, we compare the absolute accuracy and numerical efficiency of our New Algorithm with those of the banded solver described in the well-known book *Numerical Recipes in C* [35]. However, before doing that, we can make some general remarks about the validity of the new method from the numerical point of view.

At this point, it is worth remarking that the present state of our New Algorithm for banded systems is not yet completely optimized at the low level. Therefore, it cannot be directly compared to the thoroughly optimized routines included in commonly used scientific libraries such as LAPACK [34]. This further optimization will be pursued in future works. At the current state, it is natural to compare our algorithm to an explicit, high-level, not optimized routine, such as the ones in *Numerical Recipes in C* [35], and the results here should be interpreted as a hint of the final performance when all levels of optimization are tackled.

Our New Algorithm (NA) is based on equations (26a, 26b, 26c), (30) and (33). The source code of its different versions can be found in the supplementary material. The solver of [35] (NRC) belongs to a popular family of algorithms (see, for example, [36]) which work by calculating the ξ coefficients involved in the Gaussian elimination procedure in different iterations. Both for NA and for NRC, the ξ coefficients required for the resolution result from the summation of several terms. For a given set of ξ 's, Gaussian elimination-based methods first obtain the first term of the summation of every ξ then the corresponding second terms of the summations, and so on. In contrast, our method first obtains the final value of a given ξ by calculating all the terms in the corresponding summation; then once a given ξ_{IJ} is known, it computes $\xi_{I,J+1}$, and so on.

Both the NRC Gaussian elimination method for banded systems and our New Algorithm perform the same number of operations (i.e., the same number of additions, the same number of products, etc.). However, their efficiencies are different, as is shown in sec. 7.1. We believe this is due to the way the computers which run the algorithms access the memory positions which store the variables involved in the problem. The time that modern computers take to perform a floating point operation with two variables can be much smaller than the time required to access the memory positions of these two variables. In a modern computer, an addition or product of real numbers can take of the order of 10^{-9} - 10^{-8} s. If the variables involved are stored in the cache memory, to access them can take also 10^{-9} - 10^{-8} s. If they are stored in the main memory, the access can take the order of 10^{-7} s [51, 52]. The speed to access one position of memory is given not only by the level of memory (cache, main memory, disk, etc.) where it lies, but

also by the proximity of the position of memory which was immediately read previously. Therefore, it is expected that two different algorithms will require different execution times if they access the computer memory in different ways, even if they perform the same operations with the same variables.

In the NRC Gaussian elimination procedure, a given number of floating point variables is added to each entry A_{IJ} . The same number of floating point variables has to be added to A_{IJ} to calculate ξ_{IJ} in the New Algorithm. However, the *order* it is done is different in both cases. In Gaussian elimination, one row of A times a given number is added to another row of A , and this is repeated many times. For example, for erasing the subdiagonal entries of the first column of A , the first row (times the appropriate numbers) is added to rows 2 to $m_l + 1$. Then, to erase the (new) second column, the (new) second row is added to rows 3 to $m_l + 2$, and so on. Let us consider $A_{4,3}$, without a loss of generality. A given number is added to $A_{4,3}$ when the first row is added to its lower rows; after some steps, another number is added to $A_{4,3}$, when the second row is added to its lower rows. Again after some steps, another number is added to $A_{4,3}$ (when the third row is added to its lower rows). In this procedure, the memory positions that are accessed move away from the position of $A_{4,3}$, and then they come back to it, which can be suboptimal. However, in our New Algorithm, numbers are added to a memory position (say $A_{4,3}$) only once (see 25c), making the sweeping of memory positions more efficient. Some simple tests seem to support this hypothesis. We define the following loops:

- Loop 1: (Analogous to loop of NRC-Gaussian elimination)

```

for ( $K = 0$ ,  $K < 1000000$ ,  $K++$ ) do
  for ( $I = 0$ ,  $I < m_l$ ,  $I++$ ) do
    for ( $J = 0$ ,  $J < m_u - 1$ ,  $J++$ ) do
       $A[m_l][J] += A[I][J]$ 
    end for
  end for
end for
    
```

- Loop 2: (Analogous to the loop of the New Algorithm)

```

for ( $K = 0$ ,  $K < 1000000$ ,  $K++$ ) do
  for ( $I = 0$ ,  $I < m_u - 1$ ,  $I++$ ) do
     $A[m_l][I] += A[0][I] + \dots + A[m_l - 2][I]$ 
  end for
end for
    
```

The way in which Loop 1 sweeps the memory positions is analogous to that of the Gaussian elimination method (NRC), because it adds the different numbers to a given memory position ($A[m_l][I]$) in different iterations within the intermediate loop. The way Loop 2 sweeps the memory positions is analogous to that of the New Algorithm (NA), because it adds the different numbers to a given memory position ($A[m_l][I]$) just at a stretch. If we compare the times required by their executions (using $m_u = 10$), we

find the results of table 1.

Table 1. Comparison of the execution times of simple tests. The last column corresponds to data taken from sec. 7.

m_l	t_{Loop1}	t_{Loop2}	$t_{\text{Loop1}}/t_{\text{Loop2}}$	$t_{\text{NRC}}/t_{\text{NA}}$
3	0.379	0.169	2.243	1.145
10	1.255	0.509	2.466	1.780
30	3.788	1.473	2.563	2.360

This simple test gives us a clue on how the different ways to sweep memory positions can result in rather different execution times. The comparison between $t_{\text{Loop1}}/t_{\text{Loop2}}$ and $t_{\text{NRC}}/t_{\text{NA}}$ associated with the actual NRC and NA algorithms (without pivoting, with $N = 10^6$ and $m_u = m_l$, see sec. 7) is merely qualitative. This is because the way the memory access takes place in Loop 1 is not exactly the same as the way the memory access takes place in NRC, nor is it the same for Loop 2 and NA (and also the m_u 's are different). The reason why the relative performance of NA vs. NRC decreases with m for $m > 35$ approximately can be due to the fact that in our implementation, NA uses matrices which are larger than those of NRC.

7. Numerical tests

In this section we quantitatively compare our New Algorithm with the banded solver based on Gaussian elimination of [35]. We do so by comparing the accuracy and efficiency of both algorithms for solving given systems. For the sake of generality, in the first part of this section (7.1) we use random banded systems as inputs for our tests. In the second part (7.2), we use them (plus a modified version of NA) to solve a physical problem, the calculation of the Lagrange multipliers in proteins.

7.1. Performance for generic random banded systems

In the test systems for our comparisons we imposed $m_u = m_l = m$ for the sake of simplicity. We took $n = 10^3, 10^4$ and 10^5 and $m = 3, 10, 30, 100$ and 300 for all of them in our tests. In addition to this, we took $n = 10^6$ with $m = 3, 10$ and 30 . For each given pair of values of n and m , we generated a set of 1000 random $n \times n$ banded matrices whose entries are null, except the diagonal ones and their first m neighbours on the right and on the left. The value of these entries is a random number between 500 and -500 with 6 figures. The components of the independent term (the vector b in (4)) are random numbers between 0 and 1000, also with 6 figures. We tested both algorithms (from [35] and our NA) with and without pivoting. We used PowerPC 970FX 2.2 GHz machines, and no specific optimization flags were given to the compiler apart from the basic one (g++ -o solver solver.cpp). Every point in our performance plots corresponds

to the mean of 1000 tests and, in each point, we used the same random system as the input for both algorithms.

We measured the efficiency of a given algorithm by using an average of its execution times for given banded systems. In the measurement of such execution times, we considered only the computation time; i.e., the measured execution times correspond only to the solution of the banded systems, and not to other parts of the code such as the generation of random matrices and vectors, the initialization of variables or the checking and the storage of the results. The measurements of the execution times start immediately after the initialization, and the clock is stopped immediately after the unknowns x are calculated. The concrete information on how the measurement of times was implemented can be found in the source code of the programs used for our tests, which are included in the supplementary material. As is shown there, standard C libraries were used to measure the times.

The accuracy of the algorithms was determined by measuring the error of the solutions they provide (x). We quantified the error with the following formula, which corresponds to the normalized deviation of Ax from b :

$$\text{Error} := \frac{\sum_{I=1}^n |\sum_{J=1}^n A_{IJ}x_J - b_I|}{\sum_{I=1}^n |x_I|} . \quad (39)$$

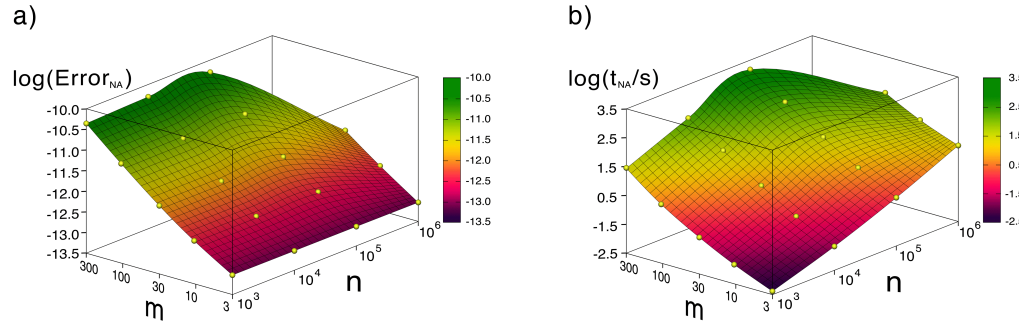


Figure 1. Properties of the New Algorithm introduced in this work, with pivoting, as a function of the size of the matrix n and the width of the band m in random banded test systems. **a)** Its accuracy, as measured by the error defined in equation (39). **b)** Its numerical efficiency, measured by the execution time.

In the first diagrams of this section (figs. 1, 2, 3 and 4) we present the absolute and relative accuracy and efficiency of the NA and NRC algorithms for the cases with and without pivoting. In these figures, the yellow spheres represent the calculated points, which correspond to the average of 1000 tests with different input random matrices and vectors. For the sake of visual confort, interpolating surfaces have been produced with cubic splines and the x and y axes (labeled n and m) are in logarithmic scale. In figures 2, 4 we compare quantities between the two algorithms; a blue plane at $z = 1$ is included. Above this plane, NA is more competitive than NRC; below this plane, the converse is true.

In figure 1a, we can see that our algorithm with pivoting has very good accuracy, with the error satisfying $\log(\text{Error}) \propto \log(m)$. The error is proportional to a power of m with a small exponent ($\simeq 1.4$). In the same figure we notice that this error is approximately independent of n . The execution time in the tested region (see figure 1b) is proportional to n and also approximately proportional to $m^{1.7}$, not to m^2 as one would expect from the number of floating point operations ($\propto nm^2$). This suggests that memory access is an important time-consuming factor, in addition to floating point operations.

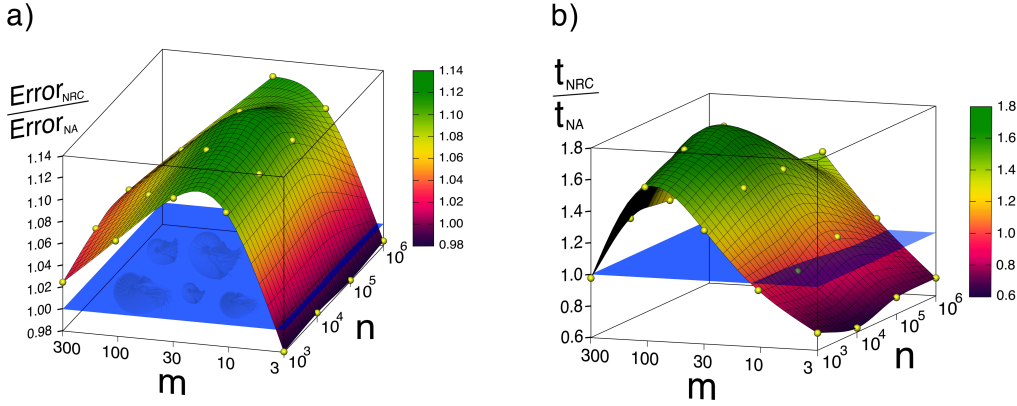


Figure 2. Comparison between the properties of the New Algorithm introduced in this work (NA) and the one in ref. [35] (NRC), both with pivoting, as a function of the size of the matrix n and the width of the band m in random banded test systems. **a)** Relative accuracy, as measured by the ratio of the errors defined in equation (39). **b)** Relative numerical efficiency, measured by the ratio of the execution times.

In figure 2a, we can see that, if pivoting is performed, our New Algorithm is always more accurate than NRC, except for a narrow range of m between 1 and 4. The typical increase in accuracy is around a 5%, reaches almost 15% for some values of n and m . In figure 2b, we can see that, if pivoting is performed, the New Algorithm is also faster than NRC for most of the studied values of n and m , with typical speedups of around 40% and the largest ones of almost 80%.

If pivoting is not performed, the accuracy decreases typically by two or three orders of magnitude (but errors still remain very low, usually around 10^{-10}). In the non-pivoting case, we also see that a few of the calculations (around 1 in 500) present errors significantly larger than the average. This probably suggests that the random procedure has produced a matrix that is close to singular with respect to the hypotheses introduced in sec. 2. In fig. 3, we show a typical example of the distribution of errors for the non-pivoting banded solvers NA and NRC in figure 3. The data corresponds to the errors of 1000 random input matrices with $n = 10^5$ and $m = 10$. In such a case, the average of the error is less representative. In this example test, the highest error in NRC is $1.57 \cdot 10^{-9}$, and in NA is $2.53 \cdot 10^{-9}$, although these numbers are probably anecdotal.

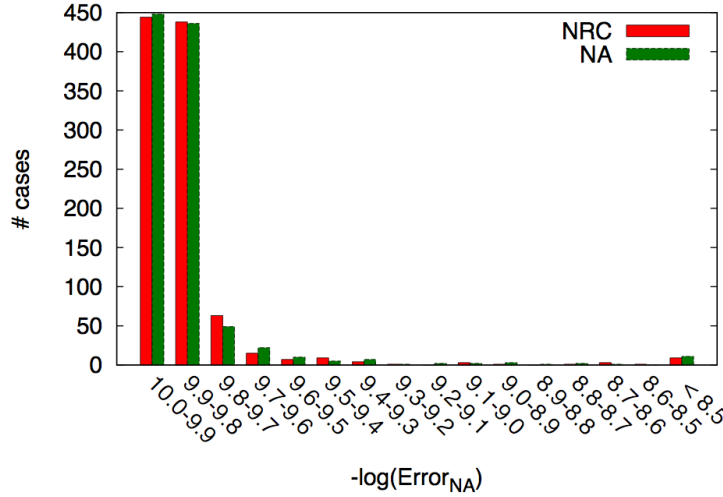


Figure 3. Histogram of the errors made by the algorithms NA and NRC for solving random banded systems without pivoting. The data corresponds to 1000 random inputs with $n = 10^5$ and $m = 10$.

One must also note that 99% of the errors are $\mathcal{O}(10^{-10})$ or smaller. A comparison of the red and green bars in the histogram suggests that there are no big differences in the errors of both algorithms (NA and NRC) without pivoting.

Despite these problems in dealing with almost singular matrices, algorithms without pivoting have an important advantage regarding computational cost, and they can be useful for problems in which the matrices are a priori known to be well behaved. These computational savings are noticed if we compare figs. 1b, and 4a. In figure 4b, we can additionally see that the New Algorithm introduced in this work is always faster than NRC for the explored values of n and m if no pivoting is performed; the increase in efficiency reaching almost to a factor of 3 for some values of n and m , and being typically around a factor of 2.

7.2. Analytical calculation of Lagrange multipliers in a protein

In Molecular Dynamics simulations, it is a common practice to *constrain* some of the internal degrees of freedom of the involved systems. This enables an increase in the simulation time step, makes the simulation more efficient, and is expected not to severely distort the value of the observable quantities calculated in the simulation [53, 54]. The bond lengths of a molecule can be constrained by including algebraic restrictions such as the following one:

$$|\vec{x}_\alpha - \vec{x}_\beta|^2 - (a_{\alpha,\beta})^2 = 0 \quad (40)$$

in the system of classical equations of motion of the atoms. In this expression, the positions of atoms in a molecule formed by N_a atoms are given by \vec{x}_α , \vec{x}_β , with $\alpha, \beta = 1, \dots, N_a$. The parameter $a_{\alpha,\beta}$ is the length of the bond which links atoms α and β .

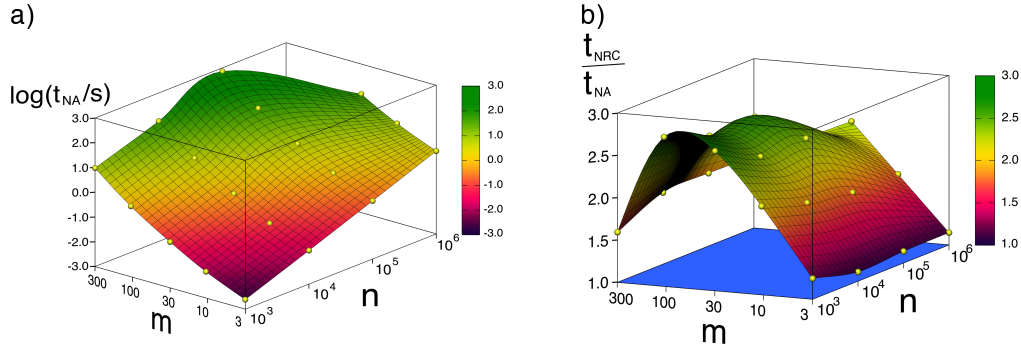


Figure 4. **a)** Numerical complexity of the New Algorithm introduced in this work, without pivoting, when solving random banded test systems. Execution time is shown as a function of the size of the matrix n and the width of the band. **b)** Comparison between the numerical complexity of NA and NRC.

The imposition of holonomic constraints such as (40) under the assumption of the D'Alembert principle makes the so-called *constraint forces* appear. These forces are proportional to their associated *Lagrange multipliers*, which have to be calculated in order to evaluate the dynamics of the system. Proteins, nucleic acids and other biological molecules have an essentially linear topology, which makes it possible to calculate the Lagrange multipliers associated to their constrained internal degrees of freedom by solving banded systems. More explanations on how to impose constraints on molecules and on how to calculate the Lagrange multipliers in biomolecules can be found in [22].

In this section, we compare the efficiencies and accuracies of three methods to solve the banded systems associated with the calculation of Lagrange multipliers of a family of relevant biological molecules (polyalanines). The three methods we compare are:

- The Gaussian elimination algorithm for banded systems presented in [35] (NRC)
- The New Algorithm (NA) presented here, based on equations (26a, 26b, 26c)
- A modified version of the New Algorithm presented here, which uses the methods discussed in sec. 3 and takes advantage in the symmetry of the system (i.e., it uses equation (27) instead of (26c))

All three methods are implemented without pivoting. The accuracies and efficiencies of the first two ones were compared in sec. 7.1 for banded matrices with random entries.

In our tests, we calculated the Lagrange multipliers of α -helix shaped polyaniline chains (as the one displayed in fig. 5) with different numbers of residues (R). See [22] for further information on the way the systems of equations to solve were generated. In our tests, we measured the error as calculated with (39), as well as the execution time of the algorithms. We ran them in a MacBook6,1 with a 2.26 GHz Intel Core 2 Duo processor.

The results are displayed in figures 6 and 7. For all the polypeptide lengths

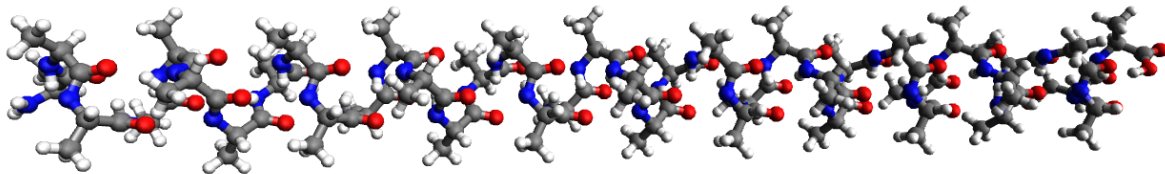


Figure 5. Polyaniline chain of 40 residues in a α -helix shape. White spheres indicate H atoms, dark spheres indicate C atoms, blue spheres indicate N atoms and red spheres indicate O atoms. The covalent bonds appear as rods connecting them. Diagram made with Avogadro [55].

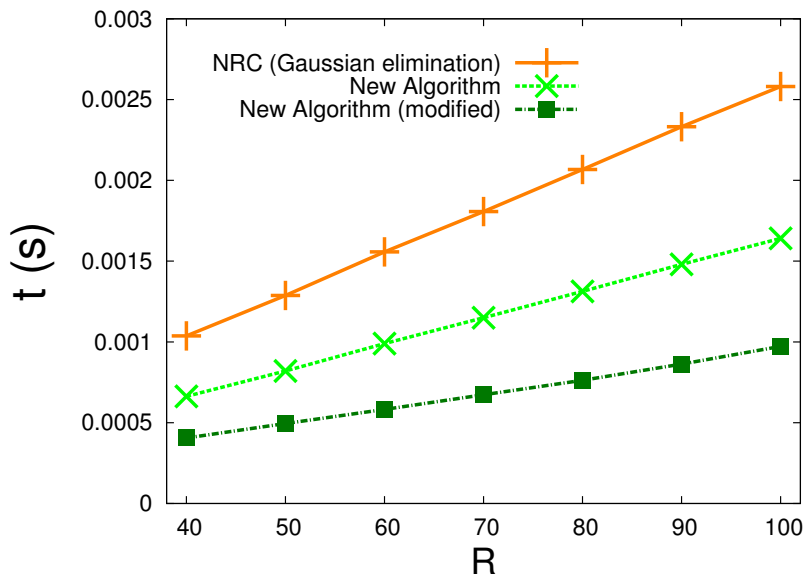


Figure 6. Comparison of the execution times (t) of different algorithms to solve banded systems in the calculation of the Lagrange multipliers in a polyaniline chain of R residues. Vertical crosses: NRC (Gaussian elimination); Diagonal crosses: NA; Squares: modified New Algorithm.

represented in fig. 6, the execution time of the Gaussian elimination algorithm (NRC) is about 1.57 times the execution time of the New Algorithm (1.57 ± 0.01). The modified New Algorithm (squares in figures 6 and 7) is about 2.70 times faster than the NRC algorithm. These results were the expected results for the used values of n , m_u and m_l ($m_u = m_l = m = 6$, $n = 10R + 2$), according to the tendencies observed in the previous section. Higher values of m are expected to result in better relative efficiency of the New Algorithm (see sec. 7.1). A situation that we can meet, for example, if not only bond lengths, but also bond angles, are constrained, and if the branches of the molecule are longer (for example, the side chains of the arginine residue are longer than the side chains of the alanine residue).

The errors made by the three tested algorithms are displayed in fig. 7. As expected

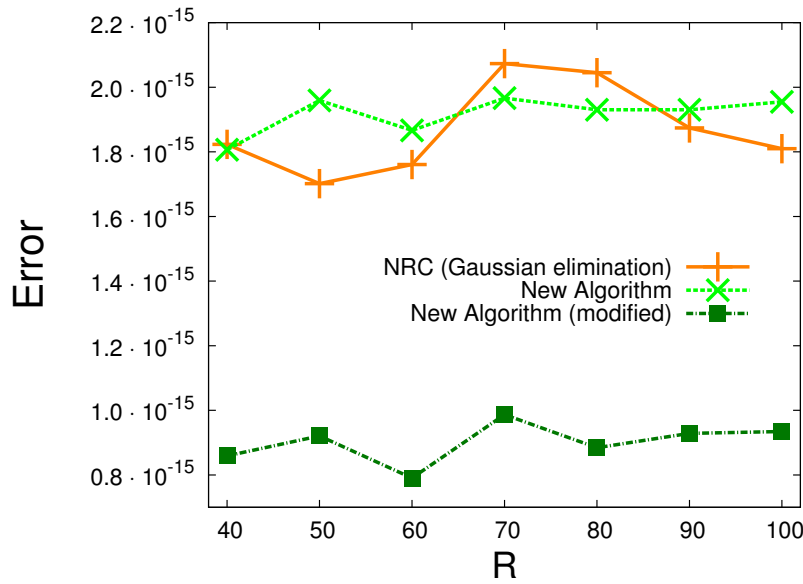


Figure 7. Comparison of the errors made by different algorithms in the solution of banded systems for the calculation of the Lagrange multipliers in a polyaniline chain of R residues. Vertical crosses: NRC (Gaussian elimination); diagonal crosses: NA; squares: modified New Algorithm.

for the case without pivoting (see sec. 7.1), the errors of the NRC and NA algorithms are similar, and both are very small (similar to the errors arisen from the finite machine precision). The error of the modified version of the New Algorithm is typically less than half of the error of the other two methods. This can be due to the fact that the modified version uses equation (27) instead of (26c). Therefore, fewer numbers (about half of them) are present in the calculation, and hence fewer potential sources of error are present.

We conclude that, for the systems tested in this section, the new algorithm introduced in this work is competitive both in accuracy and in computational efficiency when compared with a standard method for inverting banded matrices. This holds true both with and without pivoting. We stress we are comparing two algorithms which are not yet thoroughly optimized (as LAPACK is).

8. Concluding remarks

In this paper, we have introduced a new linearly scaling method to invert the banded matrices that so often appear in problems of Computational Physics, Chemistry and other disciplines. We have proven that this new algorithm is capable of being more accurate than standard methods based on Gaussian elimination at a lower computational cost, which opens the door to its use in many practical problems, such as the ones

described in the introduction.

Moreover, we have produced the analytical expressions that allow us to directly obtain, in a recursive manner, the solution to the associated linear system in terms of the entries of the original matrix. To have these explicit formulae (which have also been presented for the calculation of the full inverse matrix in the Appendix) at our disposal not only simplifies the task of coding the needed computer algorithms, but it may also be useful to facilitate analytical developments in the problems in which banded matrices appear.

In addition, we have checked its performance for general trial systems, and proven its usefulness for real physical problems (calculations on dynamics of proteins).

Aknowledgments

The authors would like to thank J. L. Alonso, G. Ciccotti, J. M. Peña, S.R. Christensen and Á. Rubio for illuminating discussions and useful advice, and M. García-Risueño for helping with the plots, as well as the staff of Caesaraugusta supercomputing facility (RES), where the test calculations of this paper were run. This work has been supported by the research projects E24/3 (DGA, Spain), FIS2009-13364-C02-01 (MICINN, Spain) 200980I064 (CSIC, Spain) and ARAID and Ibercaja grant for young researchers (Spain). P. G.-R. is supported by a JAE PREDOC grant (CSIC, Spain).

Appendix A. Inverse of a banded matrix

In the previous sections, we proved that the banded linear system of n equations with n unknowns in (1) can be solved in order n operations. Sometimes, we are interested in obtaining the inverse matrix A^{-1} itself. We can do this in order n^2 operations using the same kind of ideas discussed in the main body of the article. It should be stressed that the explicit inverse of an arbitrary banded matrix usually cannot be obtained in $\mathcal{O}(n)$ floating point operations, since the inverse of a banded matrix has n^2 entries and it is not, in general, a banded matrix itself (an exception to this is a block diagonal matrix). In order to obtain an efficient way to invert A , we will derive some recursive relations between the rows of P (and Q). To this end, we will first calculate the explicit expression of the entries of these matrices.

Using eqs. (5a), (6), and (7), from which (8a, 8b, 8c) and (28a) follow, and after some straightforward but long calculations, one can show that the aforementioned entries satisfy

$$P_{IJ} = \xi_{JJ} \sum_{c \in C_{I,J,mu}^\uparrow} \prod_{(K,L) \in c} \xi_{KL} \text{ for } I < J, \quad (1.1a)$$

$$P_{II} = \xi_{II}, \quad (1.1b)$$

$$P_{IJ} = 0 \quad \text{for } I > J, \quad (1.1c)$$

and

$$Q_{IJ} = \sum_{c \in C_{I,J,m_l}^\downarrow} \prod_{(K,L) \in c} \xi_{KL} \text{ for } I > J, \quad (1.2a)$$

$$Q_{II} = 1, \quad (1.2b)$$

$$Q_{IJ} = 0 \quad \text{for } I < J. \quad (1.2c)$$

We call the summations appearing in the first line of each of these groups of expressions *jump summations*. There are two jump summations here, one from I to J with increasing indices (\uparrow) and m_u neighbours, and a *jump summation* from I to J with decreasing indices (\downarrow) and m_l neighbours, respectively. The jump summation provides us with an explicit expression for all entries in A^{-1} , without the need to recursively refer to other entries. This can be useful in order to parallelize its calculation.

As it can be seen in the expressions, that each product in the sums contains a number of coefficients ξ_{KL} . The pairs of indices (K, L) which are included in a given product are taken from a set c . In turn, each term of the sum corresponds to a different set of pairs of indices c drawn from a set of sets of pairs of indices C_{I,J,m_u}^\uparrow (in the case of P_{IJ}) or C_{I,J,m_l}^\downarrow (in the case of Q_{IJ}). Therefore, the only detail that remains to understand these ‘jump summations’ is to specify which are the elements of these latter sets.

A given element c of either C_{I,J,m_u}^\uparrow or C_{I,J,m_l}^\downarrow can be expressed as

$$c = \{(K_1, L_1), (K_2, L_2), \dots, (K_S, L_S)\}, \quad (1.3)$$

in such a way that C_{I,J,m_u}^\uparrow comprises all possible c ’s that comply with a number of rules:

- $K_1 = I$, and $L_S = J$.
- $K_r < L_r$, for $r = 1, \dots, S$.
- $K_{r+1} = L_r$, for $r = 1, \dots, S$.
- $L_r - K_r \leq m_u$, for $r = 1, \dots, S$.

Let us see an example:

$$\sum_{c \in C_{3,6,2}^\uparrow} \prod_{(K,L) \in c} \xi_{KL} = \xi_{3,4}\xi_{4,5}\xi_{5,6} + \xi_{3,4}\xi_{4,6} + \xi_{3,5}\xi_{5,6}. \quad (1.4)$$

The rules to determine the elements of C_{I,J,m_l}^\downarrow are analogous to the ones above but they take into account that the indices decrease:

- $K_1 = I$, and $L_S = J$.
- $K_r > L_r$, for $r = 1, \dots, S$.
- $K_{r+1} = L_r$, for $r = 1, \dots, S$.
- $K_r - L_r \leq m_l$, for $r = 1, \dots, S$.

An example would be:

$$\begin{aligned} \sum_{c \in C_{5,1,3}^\downarrow} \prod_{(K,L) \in c} \xi_{KL} = & \xi_{5,4}\xi_{4,3}\xi_{3,2}\xi_{2,1} + \xi_{5,3}\xi_{3,2}\xi_{2,1} + \xi_{5,4}\xi_{4,2}\xi_{2,1} \\ & + \xi_{5,4}\xi_{4,3}\xi_{3,1} + \xi_{5,3}\xi_{3,1} + \xi_{5,2}\xi_{2,1} + \xi_{5,4}\xi_{4,1}. \end{aligned} \quad (1.5)$$

If we first focus on P , it is easy to see that, according to the properties of the jump summation, we have

$$\begin{aligned} \sum_{c \in C_{I,J,m_u}^\uparrow} \prod_{(K,L) \in c} \xi_{KL} &= \xi_{I,I+1} \sum_{c \in C_{I+1,J,m_u}^\uparrow} \prod_{(K,L) \in c} \xi_{KL} \\ &\quad + \xi_{I,I+2} \sum_{c \in C_{I+2,J,m_u}^\uparrow} \prod_{(K,L) \in c} \xi_{KL} \\ &\quad + \dots + \xi_{I,I+m_u} \sum_{c \in C_{I+m_u,J,m_u}^\uparrow} \prod_{(K,L) \in c} \xi_{KL} . \end{aligned} \quad (1.6)$$

If we insert a multiplicative factor ξ_{JJ} at both sides and use (1.1a), this equation becomes equation (32a) obtained in sec. 2:

$$P_{IJ} = \sum_{K=I+1}^{\min\{I+m_u, n\}} \xi_{IK} P_{KJ} \quad \text{for } I < J . \quad (1.7)$$

An analogous expression for Q can be obtained in a similar way:

$$Q_{IJ} = \sum_{L=J+1}^{\min\{J+m_l, n\}} Q_{IL} \xi_{LJ} \quad \text{for } I > J . \quad (1.8)$$

Now, if we define $\mu_1 := \min\{m_u, J - I\}$, and $\mu_2 := \min\{m_l, I - J\}$, since $A^{-1} = PQ$ (see (4)), we have that

$$\begin{aligned} (A^{-1})_{IJ} &= (PQ)_{IJ} = \sum_{K=1}^n P_{IK} Q_{KJ} = \sum_{K=J}^n P_{IK} Q_{KJ} = \\ &= \sum_{K=J}^n \left(\sum_{L=I+1}^{I+\mu_1} P_{LK} \xi_{IL} \right) Q_{KJ} = \sum_{L=I+1}^{I+\mu_1} \xi_{IL} \left(\sum_{K=J}^n P_{LK} Q_{KJ} \right) \\ &= \sum_{L=I+1}^{I+\mu_1} \xi_{IL} (A^{-1})_{LJ} \quad \text{for } I < J , \end{aligned} \quad (1.9)$$

which is a recursive relationship for the superdiagonal entries of A^{-1} .

Performing similar computations, we have

$$(A^{-1})_{II} = \xi_{II} + \sum_{L=I+1}^{I+\mu_1} \xi_{IL} (A^{-1})_{LJ} = \xi_{II} + \sum_{L=J+1}^{J+\mu_2} \xi_{LJ} (A^{-1})_{IL} , \quad (1.10)$$

$$(A^{-1})_{IJ} = \sum_{L=J+1}^{J+\mu_2} \xi_{LJ} (A^{-1})_{IL} \quad \text{for } I > J . \quad (1.11)$$

Using the last three equations, we can easily construct an algorithm to compute A^{-1} in $\mathcal{O}(n^2)$ floating point operations. This algorithm would first calculate $(A^{-1})_{nn} = \xi_{nn}$. Then it would use (1.9) to obtain, in this order, $(A^{-1})_{n-1,n}$, $(A^{-1})_{n-2,n}$, \dots , $(A^{-1})_{1n}$. These are the superdiagonal ($I < J$) entries of the n -th column. Then, it would use (1.11) to obtain, in this order, $(A^{-1})_{n,n-1}$, $(A^{-1})_{n,n-2}$, \dots , $(A^{-1})_{n1}$, i.e., the subdiagonal ($I > J$) entries of the n -th row. Once the n row and column of A^{-1} are known,

$(A^{-1})_{n-1,n-1}$ can be obtained with (1.10). Then (1.9) and (1.11) can be used to obtain the entries of this $(n-1)$ column and row, respectively. When calculating the entries of a column J , i.e., ξ_{KJ} with $K < J$, ξ_{LJ} is always obtained before $\xi_{L-1,J}$. When calculating the entries of a row I , i.e., ξ_{IK} with $I > K$, ξ_{IK} is always obtained before $\xi_{I,K-1}$. This procedure can be repeated for all rows and columns of A^{-1} , and the calculation of the K -th row and column can be performed in parallel.

References

- [1] Dongarra J and Johnson S L 1987 *Parallel Computing* **5** 219–246
- [2] Hyman J, Morel J, Shashkov M and Steinberg S 2002 *Computational Geosciences* **6** 333–352
- [3] Shaw R E and Garey L E 1997 *International Journal of Computer Mathematics* **65**, **1-2** 121–129
- [4] Paprzycki M and Gladwell I 1991 *Parallel Computing* **17** 133–153
- [5] Wright S J 1992 *SIAM J. Sci. Stat. Comput.* **13** 742–764
- [6] Briley W R and McDonald H 1977 *JCOP* **24**, **4** 372–397
- [7] Ariel P D 1992 *Acta Mechanica* **103** 31–43
- [8] Haddad O M, Al-Nimr M A and Shatnawi G H 2008 *Selected Papers from the WSEAS Conferences in Spain, September 2008 Santander, Cantabria, Spain*
- [9] Haddad O, Abuzaid M and Al-Nimr M 2004 *Entropy* **6**, **(5)** 413–416
- [10] Polizzi E and Shameh A H 2006 *Parallel Computing* **32** 177–194
- [11] Polizzi E and Ben Abdallah N 2004 *Journal of Computational Physics* **202**, **1** 150–180
- [12] Lumsdaine A, White J, Webber D and Sangiovanni-Vincentelli A 1988 *Research Laboratory of Electronics Dept. of Electrical Engineering and Computer Science Massachusetts Institute of Technology Cambridge, MA 02139, CH2657-5/88/0000/0308 01.000 1988IEEE*
- [13] Sanz-Serna J M and Christie I 1986 *Journal of Computational Physics* **67**, **2** 348–360
- [14] Guantes R and Farantos S C 1999 *Journal of Chemical Physics* **111**, **24** 10827–10835
- [15] Guardiola R and Ros J 1999 *Journal of Computational Physics* **111**, **24** 374–389
- [16] Castro A, Appel H, Oliveira M, Rozzi C A, Andrade X, Lorenzen F, Marques M A L, Gross E K U and Rubio A 2006 *Phys. Stat. Sol* **243** 2465
- [17] Marques M A L, Castro A, Bertsch G F and Rubio A 2003 *Comp. Phys. Comm.* **151** 60
- [18] Ryckaert J P, Ciccotti G and Berendsen H J C 1977 *J. Comput. Phys.* **23** 327–341
- [19] Alvarez-Estrada R F and Calvo G F 2004 *Journal of Physics: Condensed Matter* **16** S2037
- [20] Calvo G F and Alvarez-Estrada R F 2005 *Journal of Physics: Condensed Matter* **17** 7755
- [21] Mazars M 2007 *J. Phys. A: Math. Theor.* **40**, **8** 1747–1755
- [22] García-Risueño P, Echenique P and Alonso J L 2011 *J. Comput. Chem.* **32** 3039–3046
- [23] Strassen V 1969 *Numerische Mathematik* **13** 354–356
- [24] Alonso J L, Andrade X, Echenique P, Falceto F, Prada-Gracia D and Rubio A 2008 *Phys. Rev. Lett.* **101** 096403
- [25] Hastings W K 1970 *Biometrika* **57**, **1** 97–109
- [26] Echenique P and Alonso J L 2007 *Mol. Phys.* **105** 3057–3098
- [27] Gritsenko O V, Rubio A, Balbs L C and Alonso J A 1993 *Phys. Rev. A* **47** 1811–1816
- [28] Pearlman D A, Case D A, Caldwell J W, Ross W R, Cheatham III T E, DeBolt S, Ferguson D, Seibel G and Kollman P 1995 *Comp. Phys. Commun.* **91** 1–41
- [29] Cavasotto C N and W Orry A J May *Current Topics in Medicinal Chemistry* **7** 1006–1014
- [30] Anisimov V M and Cavasotto C N 2011 *Journal of Computational Chemistry* **32** 2254–2263
- [31] Hine N, Haynes P, Mostofi A, Skylaris C K and Payne M 2009 *Computer Physics Communications* **180** 1041 – 1053
- [32] Soler J M, Artacho E, Gale J D, García A, Junquera J, Ordejón P and Sánchez-Portal D 2002 *Journal of Physics: Condensed Matter* **14** 2745
- [33] Gillan M, Bowler D, Torralba A and Miyazaki T 2007 *Computer Physics Communications* **177** 14

- 18 proceedings of the Conference on Computational Physics 2006 - CCP 2006, Conference on Computational Physics 2006
- [34] Anderson E, Bai Z and Bischof C e a (1999) *LAPACK User's Guide* release 3.0 ed (Philadelphia: SIAM)
- [35] Press W H, Teukolsky S A, Vetterling W T and Flannery B P (2007) *Numerical recipes. The art of scientific computing* 3rd ed (New York: Cambridge University Press)
- [36] Wakins D S 1991 *Fundamentals of Matrix Computations* 2nd ed (New York: Wiley Inter-Science)
- [37] Golub G H and Van Loan C F (eds) 1993 *Matrix Computations* 2nd ed (Baltimore and London: The Johns Hopkins University Press)
- [38] Castro A, Marques M A L and Rubio A 2004 *J. Chem. Phys.* **121** 3425–3433
- [39] Chandrasekaran S and Gu M 2003 *SIAM Journal on Matrix Analysis and its Applications* **25**(2) 373–384
- [40] Bini D A and Meini B 1999 *SIAM J. Matrix Anal. Appl.* **20** 700–719
- [41] Meier U 1985 *Parallel Computing* **2** 33–23
- [42] Zhang H and Moss W F 1994 *Parallel Computing* **20**, **8** 1089–1105
- [43] Lawrie D H and Sameh A H 1984 *ACM Transactions on Mathematical Software* **10**, **2** 185–195
- [44] Johnson S L 1985 *ACM Transactions on Mathematical Software* **11** 271–288
- [45] Chen S C, Kuck D J and Sameh A H 1978 *ACM Transactions on Mathematical Software* **4** 270–277
- [46] Evans D J and Hatzopoulos M 1976 *The Computer Journal* **19**, **2** 184–187
- [47] Garey L E and Shaw R E 2000 *Applied Mathematics and Computation* **5**, **311** 133–143
- [48] Arbenz P and Gander W 1994 *Technical Report TR 221, Inst. for Scientific Comp., ETH, Zürich*
- [49] Golub G H, Sameh A H and Sarin V 2001 *Numerical linear algebra with applications* **8** 297–316
- [50] Dongarra J and Sameh A H 1984 *Parallel Computing* **1** 223–235
- [51] Hennessy J L and Patterson D A 2003 *Computer Architecture, A Quantitative Approach* 3rd ed (San Mateo, CA: Morgan Kaufmann - Elsevier)
- [52] Hager G and Wellein G 2011 *Introduction to High Performance Computing for Scientists and Engineers* 1st ed (CRC Press - Taylor & Francis Group)
- [53] Leimkuhler B and Reich S 2004 *Simulating Hamiltonian dynamics* 1st ed (Cambridge University Press - Cambridge monographs on applied and computational Mathematics)
- [54] Hess B, Bekker H, Berendsen H J C and Fraaije J G E M 1997 *J. Comput. Chem.* **18** 1463–1472
- [55] 2010 Avogadro: an open-source molecular builder and visualization tool. version 1.0.1
<http://avogadro.openmolecules.net/>